Lecture7(Part1)
Chapter8

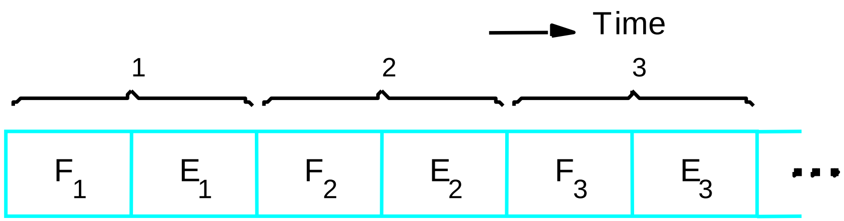
Topics covered:
Pipelining

Basic concepts

- Speed of execution of programs can be improved in two ways:
  - Faster circuit technology to build the processor and the memory.
  - Arrange the hardware so that a number of operations can be performed simultaneously. The number of operations performed per second is increased although the elapsed time needed to perform any one operation is not changed.
- Pipelining is an effective way of organizing concurrent activity in a computer system to improve the speed of execution of programs.

Basic concepts (contd..)-sequential execution

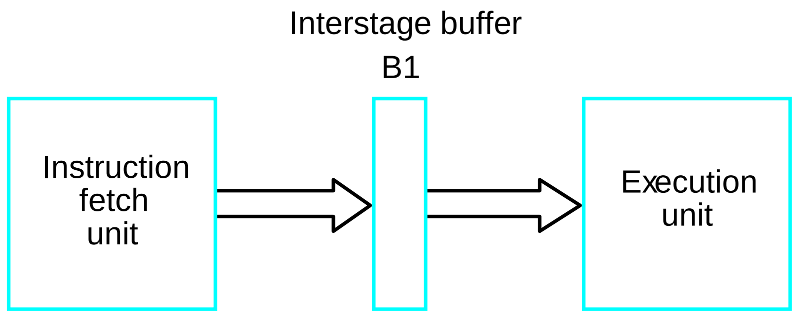
- *Processor executes a program by fetching and executing instructions one after the other.*
- *This is known as sequential execution.*
- *If F_i refers to the fetch step, and E_i refers to the execution step of instruction I_i, then sequential execution looks like:*



What if the execution of one instruction is overlapped with the fetching of the next one?

Basic concepts (contd..)-pipelined execution

- *Computer has two separate hardware units, one for fetching instructions and one for executing instructions.*
- *Instruction is fetched by instruction fetch unit and deposited in an intermediate buffer B1.*
- *Buffer enables the instruction execution unit to execute the instruction while the fetch unit is fetching the next instruction.*
- *Results of the execution are deposited in the destination location specified by the instruction.*

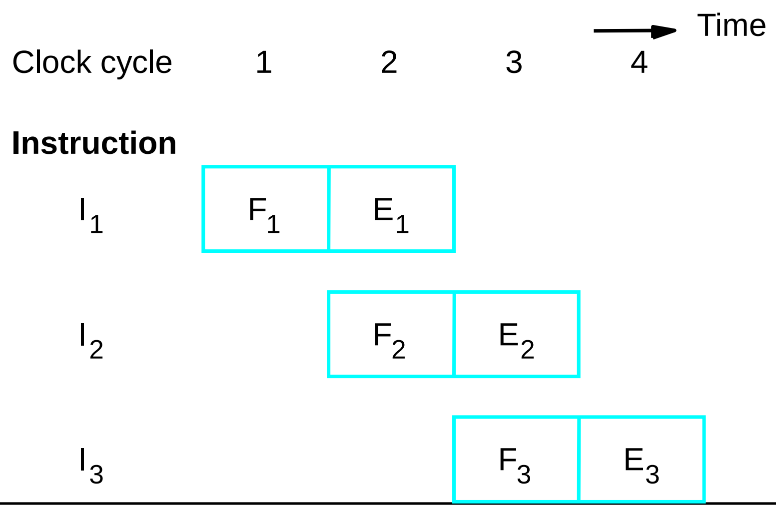


Two stages pipeline

Basic concepts (contd..)-pipelined execution

- *Computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can be completed in one clock cycle.*
- *First clock cycle:*
  - *Fetch unit fetches an instruction I_1 (F_1) and stores it in B1.*
- *Second clock cycle:*
  - *Fetch unit fetches an instruction I_2 (F_2) , and execution unit executes instruction I_1 (E_1).*
- *Third clock cycle:*
  - *Fetch unit fetches an instruction I_3 (F_3), and execution unit executes instruction I_2 (E_2).*
- *Fourth clock cycle:*
  - *Execution unit executes instruction I_3 (E_3).*

Two stages pipeline



Basic concepts (contd..)

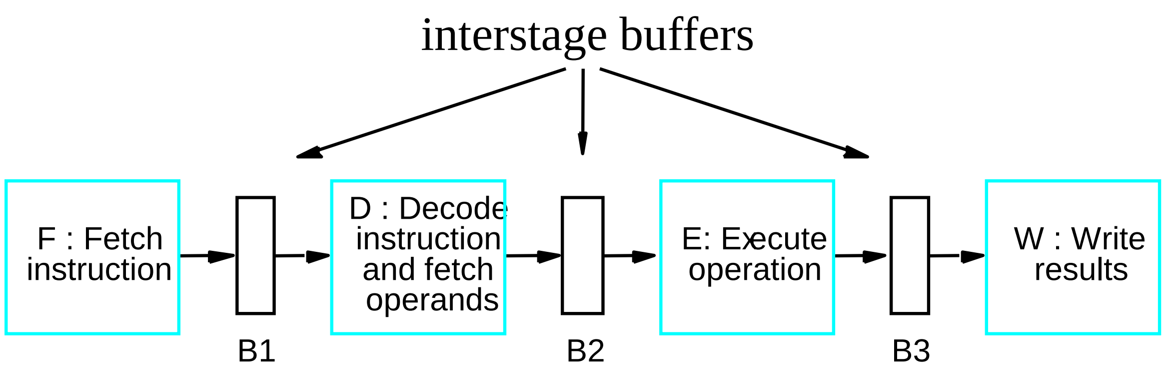
- In each clock cycle, the fetch unit fetches the next instruction, while the execution unit executes the current instruction stored in the interstage buffer.
  - Fetch and the execute units can be kept busy all the time.
- If this pattern of fetch and execute can be sustained for a long time, the completion rate of instruction execution will be twice that achievable by the sequential operation.
- Fetch and execute units constitute a two-stage pipeline.
  - Each stage performs one step in processing of an instruction.
  - Interstage storage buffer holds the information that needs to be passed from the fetch stage to execute stage.
  - New information gets loaded into the buffer every clock cycle.

pipelining doesn't decrease the instruction latency (the time for processing a single instruction); it only increases the instruction throughput (the number of instructions that can be executed in a unit of time) of the processor when processing a set of instructions.

#Give an example of pipelined machine in your environment.

Basic concepts (contd..)-four stages pipeline

- *Suppose the processing of an instruction is divided into four steps:*
  - *F Fetch: Read the instruction from the memory.*
  - *D Decode: Decode the instruction and fetch the source operands.*
  - *E Execute: Perform the operation specified by the instruction.*
  - *W Write: Store the result in the destination location.*
- *There are four distinct hardware units, for each one of the steps.*
- *Information is passed from one unit to the next through an interstage buffer.*
- *Three interstage buffers connecting four units.*
- *As an instruction progresses through the pipeline, the information needed by the downstream units must be passed along.*



Four stages pipeline

Basic concepts (contd..)-four stages pipeline

Time →

Clock cycle	1	2	3	4	5	6	7
Instruction							
I_1	F_1	D_1	E_1	W_1			
I_2		F_2	D_2	E_2	W_2		
I_3			F_3	D_3	E_3	W_3	
I_4				F_4	D_4	E_4	W_4

Clock cycle 1: F1
Clock cycle 2: D1, F2
Clock cycle 3: E1, D2, F3
Clock cycle 4: W1, E2, D3, F4
Clock cycle 5: W2, E3, D4
Clock cycle 6: W3, E3, D4
Clock cycle 7: W4

Basic concepts (contd..)

Time →

Clock cycle	1	2	3	4	5	6	7
Instruction							
I_1	F_1	D_1	E_1	W_1			
I_2		F_2	D_2	E_2	W_2		
I_3			F_3	D_3	E_3	W_3	
I_4				F_4	D_4	E_4	W_4

<u>During clock cycle #4:</u>

- *Buffer B1 holds instruction I_3, which is being decoded by the instruction-decoding unit. Instruction I_3 was fetched in cycle 3.*
- *Buffer B2 holds the source and destination operands for instruction I_2. It also holds the information needed for the Write step (W_2) of instruction I_2. This information will be passed to the stage W in the following clock cycle.*
- *Buffer B3 holds the results produced by the execution unit and the destination information for instruction I_1.*

Role of cache memory

- Each stage in the pipeline is expected to complete its operation in one clock cycle:
  - Clock period should be sufficient to complete the longest task.
  - Units which complete the tasks early remain idle for the remaining clock period.
  - Tasks being performed in different stages should require about the same amount of time for pipelining to be effective.
- If instructions are to be fetched from the main memory, the instruction fetch stage would take as much as ten times greater than the other stage operations inside the processor.
- However, if instructions are to be fetched from the cache memory which is on the processor chip, the time required to fetch the instruction would be more or less similar to the time required for other basic operations.

Pipeline performance

- Potential increase in performance achieved by using pipelining is proportional to the number of pipeline stages.
  - For example, if the number of pipeline stages is 4, then the rate of instruction processing is 4 times that of sequential execution of instructions.
  - Pipelining does not cause a single instruction to be executed faster, it is the throughput that increases.
- This rate can be achieved only if the pipelined operation can be sustained without interruption through program instruction.
- If a pipelined operation cannot be sustained without interruption, the pipeline is said to "stall".
- A condition that causes the pipeline to stall is called a "hazard".

Hazard types

- Data hazard

is a condition in which either the source or the destination operand is not available at the time expected in the pipeline

- Control or instruction hazard

is a condition in which the instruction is not available at the time expected in the pipeline

- Structural hazard

Occurs when two instructions require the use of a hardware resource at the same time(ex. Memory accessing).

Data hazard

- *Execution of the instruction occurs in the E stage of the pipeline.*
- *Execution of most arithmetic and logic operations would take only one clock cycle.*
- *However, some operations such as division would take more time to complete.*
- *For example, the operation specified in instruction I2 takes three cycles to complete from cycle 4 to cycle 6.*

→ Time

Clock cycle	1	2	3	4	5	6	7
Instruction							
I_1	F_1	D_1	E_1	W_1			
I_2		F_2	D_2	E_2	W_2		
I_3			F_3	D_3	E_3	W_3	
I_4				F_4	D_4	E_4	W_4

Data hazard (contd..)

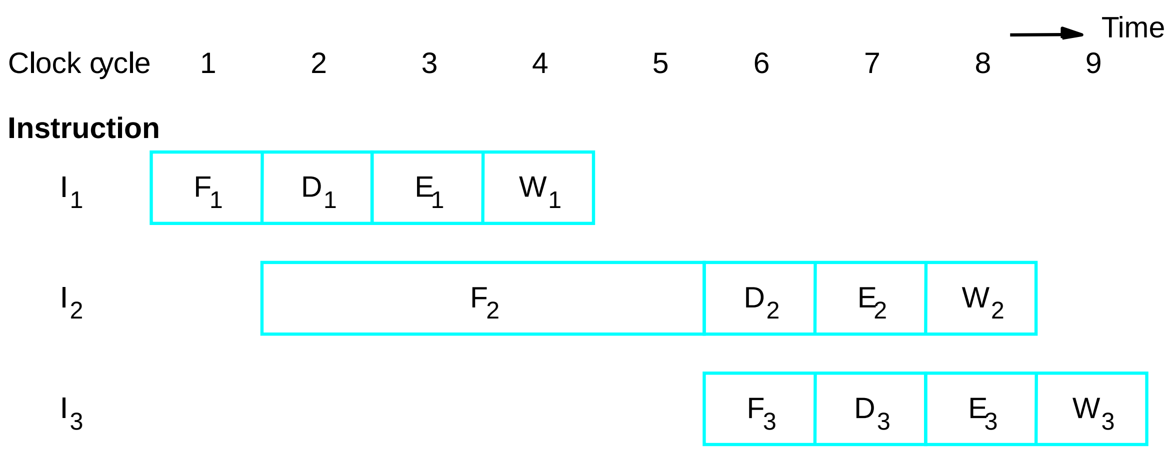
Clock cycle	1	2	3	4	5	6	7
Instruction							
I_1	F_1	D_1	E_1	W_1			
I_2		F_2	D_2	E_2	W_2		
I_3			F_3	D_3	E_3	W_3	
I_4				F_4	D_4	E_4	W_4

$\longrightarrow$ Time

- *Cycles 5 and 6, the Write stage is idle, because it has no data to work with.*
- *Information in buffer B2 must be retained till the execution of the instruction I_2 is complete.*
- *Stage 2, and by extension stage 1 cannot accept new instructions because the information in B1 cannot be overwritten.*
- *Steps D_6 and F_5 must be postponed.*
- *A data hazard is a condition in which either the source or the destination operand is not available at the time expected in the pipeline.*

Control or instruction hazard

- *Pipeline may be stalled because an instruction is not available at the expected time.*
- *For example, while fetching an instruction a cache miss may occur, and hence the instruction may have to be fetched from the main memory.*
- *Fetching the instruction from the main memory takes much longer than fetching the instruction from the cache.*
- *Thus, the fetch cycle of the instruction cannot be completed in one cycle.*
- *For example, the fetching of instruction I_2 results in a cache miss.*
- *Thus, F_2 takes 4 clock cycles instead of 1.*



Control or instruction hazard (contd..)

- *Fetch operation for instruction I2 results in a cache miss, and the instruction fetch unit must fetch this instruction from the main memory.*
- *Suppose fetching instruction I2 from the main memory takes 4 clock cycles.*
- *Instruction I2 will be available in buffer B1 at the end of clock cycle 5.*
- *The pipeline resumes its normal operation at this point.*
- *Decode unit is idle in cycles 3 through 5.*
- *Execute unit is idle in cycles 4 through 6.*
- *Write unit is idle in cycles 5 through 7.*
- *Such idle periods are called as stalls or bubbles.*
- *Once created in one of the pipeline stages, a bubble moves downstream unit it reaches the last unit.*

→ Time

Clock cycle	1	2	3	4	5	6	7	8	9
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2	F_3			
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W: Write				W_1	idle	idle	idle	W_2	W_3

Structural hazard

- Two instructions require the use of a hardware resource at the same time.
- Most common case is in access to the memory:
  - One instruction needs to access the memory as part of the Execute or Write stage.
  - Other instruction is being fetched.
  - If instructions and data reside in the same cache unit, only one instruction can proceed and the other is delayed.
- Many processors have separate data and instruction caches to avoid this delay.
- In general, structural hazards can be avoided by providing sufficient resources on the processor chip.

Structural hazard (contd..)

Clock cycle **Instruction**	1	2	3	4	5	6	7	
I_1	F_1	D_1	E_1	W_1				
I_2		F_2	D_2	E_2	M_2	W_2		(Load X(R1),R2
I_3			F_3	D_3	E_3		W_3	
I_4				F_4	D_4		E_4	
I_5						F_5	D_5	

- *Memory address X+R1 is computed in step E_2 in cycle 4, memory access takes place in cycle 5, operand read from the memory is written into register R2 in cycle 6.*
- *Execution of instruction I_2 takes two clock cycles 4 and 5.*
- *In cycle 6, both instructions I_2 and I_3 require access to register file.*
- *Pipeline is stalled because the register file cannot handle two operations at once.*

Pipelining and performance

- Pipelining does not cause an individual instruction to be executed faster, rather, it increases the throughput.
  - Throughput is defined as the rate at which instruction execution is completed.
- When a hazard occurs, one of the stages in the pipeline cannot complete its operation in one clock cycle.
  - The pipeline stalls causing a degradation in performance.
- Performance level of one instruction completion in each clock cycle is the upper limit for the throughput that can be achieved in a pipelined processor.

Data hazards

- *Data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed.*
- *Consider two instructions:*

$$I_1: A = 3 + A$$
$$I_2: B = 4 \times A$$

- *If $A = 5$, and I_1 and I_2 are executed sequentially, $B=32$.*
- *In a pipelined processor, the execution of I_2 can begin before the execution of I_1.*
- *The value of A used in the execution of I_2 will be the original value of 5 leading to an incorrect result.*
- *Thus, instructions I_1 and I_2 depend on each other, because the data used by I_2 depends on the results generated by I_1.*
- *Results obtained using sequential execution of instructions should be the same as the results obtained from pipelined execution.*
- *When two instructions depend on each other, they must be performed in the correct order.*

Data hazards (contd..)

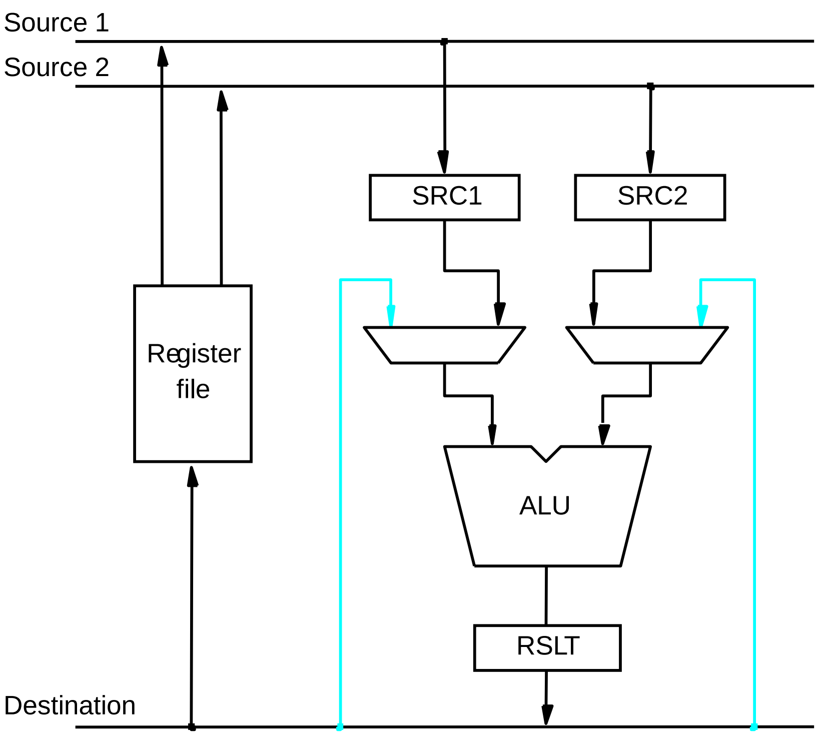
Clock cycle / **Instruction**	1	2	3	4	5	6	7	8	9	
I_1	F_1	D_1	E_1	W_1						Mul R2, R3, R4
I_2		F_2	D_2		D_{2A}	E_2	W_2			Add R5,R4,R6
I_3			F_3			D_3	E_3	W_3		
I_4						F_4	D_4	E_4	W_4	

- *Mul instruction places the results of the multiply operation in register R4 at the end of clock cycle 4.*
- *Register R4 is used as a source operand in the Add instruction. Hence the Decode Unit decoding the Add instruction cannot proceed until the Write step of the first instruction is complete.*
- *Data dependency arises because the destination of one instruction is used as a source in the next instruction.*

Operand forwarding(data hazard solution)

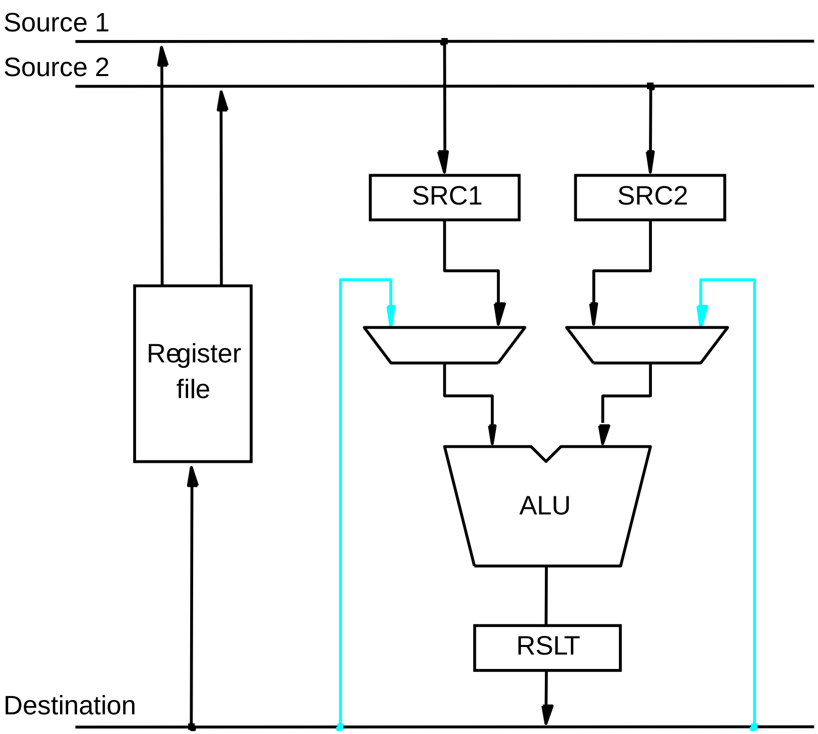
- *Data hazard occurs because the destination of one instruction is used as the source in the next instruction.*
- *Hence, instruction I_2 has to wait for the data to be written in the register file by the Write stage at the end of step W_1.*
- *However, these data are available at the output of the ALU once the Execute stage completes step E_1.*
- *Delay can be reduced or even eliminated if the result of instruction I1 can be forwarded directly for use in step E_2.*
- *This is called "operand forwarding".*

Operand forwarding (contd..)



- *Similar to the three-bus organization.*
- *Registers SRC1, SRC2 and RSLT have been added.*
- *SRC1, SRC2 and RSLT are interstage buffers for pipelined operation.*
- *SRC1 and SRC2 are part of buffer B2.*
- *RSLT is part of buffer B3.*
- *Data forwarding mechanism is shown by the two red lines.*
- *Two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of SRC1 and SRC2 register.*

Operand forwarding (contd..)



I_1: Mul R2, R3, R4
I_2: Add R5, R4, R6

Clock cycle 3:

- *Instruction I_2 is decoded, and a data dependency is detected.*
- *Operand not involved in the dependency, register R5 is loaded in register SRC1.*

Clock cycle 4:

- *Product produced by I_1 is available in register RSLT.*
- *The forwarding connection allows the result to be used in step E_2.*

Instruction I_2 proceeds without interruption.

Handling data dependency in software

- Data dependency may be detected by the hardware while decoding the instruction:
  - Control hardware may delay by an appropriate number of clock cycles reading of a register till its contents become available. The pipeline stalls for that many number of clock cycles.
- Detecting data dependencies and handling them can also be accomplished in software.
  - Compiler can introduce the necessary delay by introducing an appropriate number of NOP instructions. For example, if a two-cycle delay is needed between two instructions then two NOP instructions can be introduced between the two instructions.

I_1: Mul R2, R3, R4
NOP
NOP
I_2: Add R5, R4, R6

Side effects

- Data dependencies are explicit easy to detect if a register specified as the destination in one instruction is used as a source in the subsequent instruction.
- However, some instructions also modify registers that are not specified as the destination.
  - For example, in the autoincrement and autodecrement addressing mode, the source register is modified as well.
- When a location other than the one explicitly specified in the instruction as a destination location is affected, the instruction is said to have a "<u>side effect</u>".
- Another example of a side effect is condition code flags which implicitly record the results of the previous instruction, and these results may be used in the subsequent instruction.

Side effects (contd..)

I_1: Add R3, R4
I_2: AddWithCarry R2, R4

Instruction I_1 sets the carry flag and instruction I_2 uses the carry flag leading to an implicit dependency between the two instructions.

- *Instructions with side effects can lead to multiple data dependencies.*
- *Results in a significant increase in the complexity of hardware or software needed to handle the dependencies.*
- *Side effects should be kept to a minimum in instruction sets designed for execution on pipelined hardware.*